2010 12th International Conference on Computer Modelling and Simulation

A Stackelberg Game for Modelling Asymmetric Users' Behavior in Grid Scheduling

Joanna Kołodziej
Department of Mathematics and Computer Science
University of Bielsko-Biała
ul. Willowa 2, Bielsko-Biała, Poland
Email: jkolodziej@ath.bielsko.pl

Fatos Xhafa
Department of Computer Science and Information Systems
Birkbeck, University of London
Malet Street, Bloomsbury,
London WC1E 7HX, UK.
Email: fatos@dcs.bbk.ac.uk

Marcin Bogdański
Faculty of Mechanical Engineering and Computer Science
University of Bielsko-Biała
ul. Willowa 2, Bielsko-Biała, Poland
Email: marcin.bogdanski@gmail.com

***Abstract*—In traditional distributed computing the users and owners of the computational resources usually belong to the same administrative domain. Therefore all users are equally entitled to use the resources. The situation is completely different in large-scale emergent distributed computing systems, such as Grid systems, where the roles of the users are asymmetric as regards their access rights and usage of resources. Further, unlike traditional distributed computing case, Grid systems introduce hierarchical levels, which are to be taken into account for optimizing the overall system's performance. In this paper we present a Stackelberg game for modelling asymmetric users' behavior in Grid scheduling scenario. We define a two-level game with a Leader at the first level and the rest of users, called Followers, at the second one. The Leader is responsible for computing a planning of his tasks, which is usually a large fraction of the total pool of tasks in the batch. The Followers try to select the best strategy for the assignments of their tasks subject to Leader's strategy. The Stackelberg game is then translated into a hierarchical optimization problem, which is solved by Genetic Algorithm (GA) on the Leader's level and by ad hoc heuristic combined with GA on the Followers' level. We have experimentally evaluated the approach through a benchmark of static instances and report computational results for resource utilization, makespan and flowtime.**

***Keywords*-Computational Grid, Stackelberg game, Scheduling, Makespan, Flowtime, Resource utilization.**

I. Introduction

Grid systems are large-scale distributed systems built up using resources from different institutions, enterprises and/or particular owners. The cross-domain is thus one of the most important features of Grid systems. Due to this feature, the Grid users have different access rights to resources and they stand in an asymmetric position with regard to resource usage privileges. The asymmetric behavior of Grid users directly impacts the scheduling process.

In this paper we define a Stackelberg game to model the Grid users' behavior. We use this model for finding near-optimal solutions of the problem of independent batch scheduling in Grid systems. The Stackelberg games have been well-studied in the game theory literature (see, e.g. [Baçsar and Olsder, 1995]). Roughgarden [Roughgarden, 2001] defined a Stackelberg game model for scheduling tasks on a set of machines with load-dependent latencies in order to minimize the total latency of the system.

In our Stackelberg game scenario, one of the users acts as a *Leader* and the rest are his *Followers*. The Leader may hold his strategy fixed while the followers react independently subject to the Leader's strategy. There are actually many real-life Grid scenarios in which this game model can characterize situations where there is a Leader who acts first. Some such scenarios can be defined as follows:

- There is a privileged user (Leader), who can have a full access to resources as opposed to the other users who can be granted only limited access to resources.
- Considering a pool of tasks, the Leader could be the owner of a large portion of the tasks in the pool; it might then seem reasonable to allocate his tasks in the best resources in the system.
- Some tasks could have critical deadlines (for example in real-time scheduling). Then a Leader would first send the information of these tasks to the meta-broker requesting to allocate them first.
- Some tasks could have security requirements. Then the Leader can send the information of these tasks to the meta-broker requesting to allocate them in the most trustful resources (secure machines).
- Tasks arriving at a Grid system could be disparate in their needs for computational resources. Some of them could be atomic tasks generated by compound tasks while the others could be just monolithic applications. The high degree of heterogeneity of tasks is the crucial factor conditioning the Grid system's performance. In such a scenario the Leader could create a small batch of the most time consuming tasks, out the task pool, in order to "balance" the disparity. These tasks would then be sent to the meta-broker requesting to allocate them first.

It can be seen that the selection of the Leader's initial strategy in the game is crucial for the Followers' action. We focus in this paper on the second scenario presented above.

The users' Stackelberg game can be analyzed as a two-

978-0-7695-4016-0/10 $26.00 © 2010 IEEE
DOI 10.1109/UKSIM.2010.97

level hierarchical optimization problem where the Leader optimizes his cost function based on Followers' rational behavior. For solving the game we have designed three hybrid meta-heuristics combining Genetic Algorithms (GA) at Leader's and Followers'levels and three well-known ad-hoc scheduling methods at Followers' level, namely *Min-Min*, *Max-min* and *Sufferage*. The proposed game model has been then evaluated through a static benchmark of instances and three basic metrics, namely average resource utilization, makespan and flowtime are evaluated.

The remainder of this paper is structured as follows. In Sect. II we define a Grid meta-broker model and recall some preliminary concepts of independent task batch scheduling. We define the game model and describe the Stackelberg game scenario for Grid users in Sect. III. In Sect. IV three hybrid GA-based schedulers for solving the users' game are introduced. A preliminary experimental evaluation using a static benchmark is presented in Sect. V. In Sect.VI we conclude the paper and discuss future research directions.

II. Preliminaries

A. System model

Computational Grid, as a large-scale distributed computing platform, is usually modelled as a hierarchical system for an effective management of tasks and resources. In [Kwok et al., 2007] the hierarchy consists of three levels: the global scheduling level, the inter-site level, and the intra-site level. In the Meta-broker models (see e.g. [Garg et al., 2009]) just two levels are defined: there is a centralized job manager (meta-broker) responsible for controlling the resource allocation and communication between consumers (users), and service providers (resource owners). In this work we adapted this model to define a possible Grid infrastructure where the users play asymmetric roles in their community. One of the users is acting as a Leader, who usually controls a significant fraction of the task pool over time and can define (and send to the meta-broker as the requests) some preferences in scheduling. The remaining users, the Followers, follow the Leader's action and try to select strategies to optimize their game cost functions.

A general idea of a simple meta-broker model of Grid site with the Leader in user's community is presented in Fig. 1. In this model there are several independent users (Followers in a game scenario) submitting their own tasks to the Grid site. Note that we assume here the centralized version of the scheduling. In the decentralized version there are defined different groups of users , each of them having its own Leader.

We consider in this work the Independent Task Scheduling problem, in which tasks are processed in batch mode [Xhafa et al., 2007 c)]. For modelling the scheduling problem, we use the *Expected Time to Compute* model [Ali et al., 2000] with the following instance data, which has to be provided:

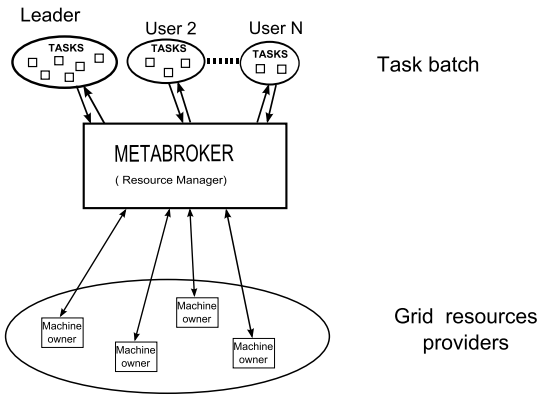

Figure 1. The meta-broker model of Grid site with one user acting as a Leader and the rest of users as its Followers.

- the estimation of computational load of each task (e.g. in millions of instructions);
- the computing capacity of each machine (e.g. in millions of instructions per second, MIPS);
- the estimation of the prior load of each available machine;
- the *ETC* matrix, in which $ETC[t][r]$ indicates the expected time to compute the task t in resource the r.

III. Stackelberg game for Grid users

An N-persons Stackelberg game can be defined as two-level game where the players act sequentially as follows. On the first level only the Leader is active, who chooses his strategy for a portion of task pool in the batch. Then, on a second level, the Followers, react rationally to the Leader's move. It means that they try to minimize their game cost functions for their own tasks subject to the Leader's choice. Finally, the Leader updates his strategy to minimize the total cost of the game.

The general Stackelberg game scenario can be used to model the asymmetric Grid users' behavior.

A. General game scenario

To define the Stackelberg game for the Grid users let us introduce the following notation:

- N is the number of users;
- n is the total number of tasks in a batch, that is, the sum of tasks submitted by all users: $n = \sum_{l=1}^{N} k_l$, where k_l is the number of tasks submitted by the user $l : l = 1, \ldots, N$;
- $\{J_1, \ldots, J_N\}; l = 1, \ldots, N$ are the sets of strategies of the users. We assume that J_1 is the set of the Leader's strategies. Note that a strategy is essentially a schedule of tasks to available machines.

- α denotes the portion of the tasks pool owned by Leader, i.e. $\alpha = card(J_1)$, where $card(X)$ denotes the number of elements of a set X;
- $x^1 = [x_1, \ldots, x_{k_1}] \in J_1$ for $k_1 = n \cdot \alpha$ is the Leader's strategy vector;
- $x^l = [x_{k_{l-1}+1}, \ldots, x_{k_l}] : l = 2, \ldots, N$ where $k_2 + \ldots + k_N = (1-\alpha) \cdot n$ are the Followers' strategy vectors;
- $\{Q_1, \ldots, Q_N\}; Q_l : J_1 \times \ldots \times J_N \rightarrow \mathbb{R}^+; \forall l = 1, \ldots, N$ is the set of users' game cost functions.

The Stackelberg game for the users could be then defined as follows:

- **Leader's Level: Leader's action I -** Leader chooses his initial strategy. His strategy vector $\widehat{x^1} = [\widehat{x_1}, \ldots, \widehat{x_{k_1}}]$ represents Leader's initial action.
- **Followers' Level: Followers' action -** Each Follower minimizes his cost function relative to the Leader's strategy:

$$\begin{cases} x_F^2 = arg\min_{x^2 \in J_2} \{Q_2(\widehat{x^1}, x^2, \ldots, x^N)\} \\ \vdots \\ x_F^N = arg\min_{x^N \in J_N} \{Q_N(\widehat{x^1}, \ldots, x^N)\} \end{cases} \quad (1)$$

  Let us denote by $x_F = [\widehat{x^1}, x_F^2, \ldots, x_F^N]$ the vector which is interpreted as the result of the Followers' action.
- **Leader's Level: Leader's action II -** Leader updates his strategy by minimizing his cost function Q_l taking into account the result of Followers' action. The following vector $x_G = [x_L, x_F^2, \ldots, x_F^N]$, where:

$$x_L = arg \min_{x^1 \in J_1} Q(x^1, x_F^2, \ldots, x_F^N) \quad (2)$$

  is a solution of the whole game.

Note that the Followers play an "ordinary" non-cooperative game, but they must know the Leader's action first. An optimal solution of the whole game is called *Stackelberg Equilibrium*.

B. Users' cost functions

The user's cost function Q_l, introduced in Section III-A, is defined on the set of all possible schedules $Sched = J_1 \times \ldots \times J_N$. A *schedule* of tasks submitted by all users at the Grid site is expressed as a vector $x = [x^1, \ldots, x^N]$, where $x^l = [x_{k_{l-1}+1}, \ldots, x_{k_l}]$ denotes an l-th user's strategy vector, in which $x_j \in [1, m]$ indicates the number of the machine, to which task j is allocated ($j = k_{l-1}+1, \ldots, k_l$).

In our approach the user cost Q_l of playing a Stackelberg game can be defined as the sum of the following two components:

$$Q_l = Q_l^{(e)} + Q_l^{(u)}, \quad (3)$$

where:

- $Q_l^{(e)}$ indicates user's task execution cost,
- $Q_l^{(u)}$ denotes resource utilization cost calculated for a given user.

Each Grid user l submits a set of k_l independent tasks. The total cost of the execution of the tasks belonged to the user l can be calculated as an average completion time of execution of all his tasks on machines, to which they are allocated. We can then define function $Q_l^{(e)}$ using the following formulae:

$$Q_l^{(e)} = \frac{\sum_{j=k_{l-1}+1}^{k_l} completion[j][x_j]}{completion_m \cdot k_l}, \quad (4)$$

where $completions[j][x_j]$ denotes the completion time of a task j on machine x_j and it is calculated in the following way:

$$completions[j][x_j] = ETC[j][x_j] + ready[x_j]. \quad (5)$$

We denoted by $completion_m$ in Eq. 4 the maximal completion time of the user's tasks, i.e :

$$completion_m = \max_{j=k_{l-1}+1,\ldots,k_l} completion[j][x_j]. \quad (6)$$

Note that in Eqs. (4) and (5) $ETC[j][x_j]$ is an element of the *ETC* matrix and $ready[x_j]$ is the finishing time of the execution of tasks previously assigned to the machine x_j.

The second component of Q_l, denoted by $Q_l^{(u)}$, refers to the utility function in Grid system, which is usually defined as the cost of the resource utilization. For instance, in [Garg et al., 2009] the utilization cost is calculated for the Grid users as the cost of buying free CPU cycles. In our approach each user tends to minimize his "portion" of a price for the idle time units of machines on which his tasks are executed. We define the $Q_l^{(u)}$ function using the following expression:

$$Q_l^{(u)} = \sum_{x_j \in machines(l)} \left(1 - \frac{Completion_{(l)}[x_j]}{loc\_makespan}\right) \cdot P[x_j] \quad (7)$$

where $machines(l)$ denotes a set of machines, to which all tasks of the user l are assigned. We denote by $loc\_makespan$ the local makespan which is defined as the maximal completion time of all machines. The completion time of the machine $x_j \in machines(l)$, denoted by $Completion_{(l)}[x_j]$, is calculated in the following way:

$$Completion_{(l)}[x_j] = ready[x_j] + \sum_{\substack{p \in \{1,\ldots,n\}: \\ x[p]=x_j}} ETC[p][x_j] \quad (8)$$

where $x[p]$ is the value of p-th coordinate in a given schedule vector x.

The fraction:

$$\frac{Completion_{(l)}[x_j]}{loc\_makespan} \quad (9)$$

in Eq. (7) is interpreted as the cost of utilization of machine x_j. The fraction of the cost of the idle time of machine x_j calculated for a given user l is defined by:

$$\left(1 - \frac{Completion_{(l)}[x_j]}{loc\_makespan}\right) \cdot P[x_j], \tag{10}$$

where:

$$P[x_j] = \frac{\sum_{j \in Tasks_{(l)}[x_j]} ETC[j][x_j]}{Completion_{(l)}[x_j]} \tag{11}$$

and $Tasks_{(l)}[x_j]$ is the set of the tasks of the user l assigned to the machine x_j. The fraction $P[x_j]$ expresses a relative time of execution of all tasks of the user l assigned to the machine x_j (with respect to the completion time of the machine x_j).

It follows from Eq. (7) that the utilization cost[1] is minimal in the case of allocation of the user's tasks to machines with the maximal completion time. It means that the main resource utilization criterion is satisfied.

IV. GA-based hybrid schedulers for solving the Grid users' game

The problem of solving the Stackelberg game can be formulated as a hierarchical two-level optimization problem. Once the Leader chooses his strategy, we have to minimize, at the Followers' level, the Followers' cost functions (see Eq. (1)). Then, the Leader's solution as specified by Eq. (2) is calculated at the Leader's level.

To solve this hierarchic optimization problem, we propose three *Hybrid GA-based Schedulers*, where GA at Leader's and Followers' levels is hybridized with three ad-hoc scheduling heuristics, namely *Min-Min*, *Max-min* and *Sufferage*. We denote the resulting hybrid algorithms by *GA-MinMin*, *Ga-MaxMin* and *GA-Sufferage*, respectively. The general template of that GA-based scheduler at the Leader's level is given in Alg. 1.

Algorithm 1 A hybridized GA-based scheduler template - Leader's level

1: Generate P^0 containing μ "incomplete" schedules; $t = 0$;
2: Send P^0 to the **Followers** to complete the respective parts of all schedules in P^0 *(using ad-hoc heuristics)*; $P^0(F)$ is created;
3: Update the population P^0 according to the Followers' solutions;$P^0 := P^0(F)$;
4: Evaluate P^0;
5: **while** not termination-condition **do**
6: Select the parental pool T^t of size λ; $T^t := Select(P^t)$;
7: Perform crossover procedures separately on Leader's and Followers'variables on pairs of individuals in $T^t(F)$ with probability p_c; $P_c^t := Cross(T^t)$;
8: Perform mutation procedures separately to Leader's and Followers' variables on individuals in P_c^t with probability p_m; $P_m^t := Mutate(P_c^t)$;
9: Evaluate P_m^t ;
10: Create a new population P^{t+1} of size μ from individuals in P^t and P_m^t ; $P^{t+1} := Replace(P^t; P_m^t)$
11: $t := t + 1$;
12: **end while**
13: **return** Best found individual as solution;

[1]The utilization cost for the user l is expressed in arbitrary time units.

The process of initialization of the population in our hybrid GA is defined as a two-steps procedure. In the first step we define P^0 as a candidate initial population. It consists of the incomplete schedules computed by the Leader using one of the initialization methods for GA-based schedulers (see e.g. [Xhafa et al., 2007 a)]). Each schedule from this set contains just the values of the Leader's decision variables. All those "incomplete" chromosomes are sent to the Followers, who fill in the respective parts of each schedule by using one of the ad-hoc heuristics. The updated P^0 is then evaluated under the game cost function $Q = \sum_{l=1...N} Q_l$ defined as the fitness. The crossover and mutation operations are performed separately on Leader's and Followers' decision variables. Thus in each generation the Followers can update their own decision (including the initial choices) due to possible changes on availability of resources introduced by the Leader.

Based on the results of the tuning process of GA operators performed in [Xhafa et al., 2007 a)] we used linear ranking selection, cycle crossover (CX) and re-balancing mutation as an appropriate combination of such operators for our algorithm. We also applied the elitist generational replacement method and *LJFR-SJFR (Longest Job to Fastest Resource – Shortest Job to Fastest Resource)* as the initialization procedure of generating P^0 defined for the Leader.

V. Experimental analysis for static scheduling

The dynamic schedulers working in the batch mode run the static methods for scheduling tasks sampled in particular batches. Thus our preliminary experimental study has been conducted on a subset of the benchmark of static instances. Those instances are classified into different types of ETC matrices, according to task heterogeneity, machine heterogeneity and consistency. Each instance consists of 512 tasks and 16 machines and is labeled by *u_x_yyzz.0* as in [Braun et al., 2001] (in the notation, hi means high, and lo means low):

- u means uniform distribution (used in generating the matrix).
- x means the type of consistency (c–consistent, i–inconsistent and s – semi-consistent).
- yy indicates the heterogeneity of the tasks.
- zz indicates the heterogeneity of the resources.

We perform our experiments for a representative group of 17 Grid users with one selected as a Leader. The Leader maintains the half of the whole task pool, i.e. 256 tasks. The rest of the tasks in the pool is uniformly distributed among 16 Followers, i.e. each of them owns 16 tasks. In this study we assume that Leader owns 50% of task pool (i.e. $\alpha = 0.5$). The settings of parameters is presented in Table I.

We set the base population size in GA on the Leader's level as 60 individuals, intermediate population size - 48, the crossover and mutation probabilities are fixed to 0.8 and 0.3,

Table I
GENERAL SIMULATION PARAMETER SETTING.

Parameter	Value setting
Total number of tasks	512
Number of machines	16
Number of users	17
Leader's fraction of task pool (α)	0.5
Number of tasks belonging to the Leader	256
Number of tasks belonging to individual Follower	16

respectively. The algorithm has been stopped after executing 2000 generations.

Performance Metrics Evaluated: We used the following three metrics for evaluating performance of the proposed schedulers:

- *Flowtime:* Let F_j denotes the time when task j finalizes. The flowtime can be calculated using the following formulae:

$$Flowtime = \sum_{j \in Task} F_j \tag{12}$$

  Flowtime is usually considered as a QoS criterion as it expresses the response time to the submitted task execution requests of Grid users.
- *Makespan:* Makespan is one of the basic metrics of a Grid systems performance: the smaller the value of makespan, the faster is the execution of tasks in the Grid system. Makespan can be calculated by the following formulae:

$$Makespan = \max_{j \in Tasks} F_j \tag{13}$$

- *Average Resource Utilization:*

$$Avg\_utilization = \frac{\sum_{i \in Machines} completion[i]}{makespan \cdot m} \tag{14}$$

  where m denotes number of machines, $machines$ is the set of all machines in a given batch and $completion[i]$ is a completion time of machine i and it is calculated adapting the formulae of Eq. (8) for the whole schedule.

The aim of an optimal scheduling is to minimize both $flowtime$ and $makespan$ and maximize the *average resource utilization* over the set of all possible schedules for a Grid configuration.

A. Computational results

We present in Table II computational results obtained for the static benchmark using the three meta-heuristic implementations, denoted as in Section IV. In order to avoid biased results, each experiment was repeated 30 times and average values of three considered metrics are reported.

Analysis of the results: By analyzing the average *flowtime* values we can observe that *GA-MinMin* outperforms other algorithms in all but three instances, for which *GA-Sufferage* works better. In the case of *makespan GA-MinMin* outperforms two remaining meta-heuristics in 6 instances, *GA-Sufferage* – in 5 instances and both metaheuristics are worse than *GA-MaxMin* just in one case of high heterogeneity of machines for the consistent ETC matrix .

We can conclude that *GA-MinMin* is the most effective in comparison with the remaining two GA-based hybrids in the optimization of the flowtime. Only in two cases of low heterogeneity of resources for the consistent ETC matrix and in one case of high heterogeneity of resources and tasks for the semi-consistent ETC, *GA-MinMin* is worse than *GA-Sufferage*, but the differences in the results are minor. In the case of makespan the efficiencies of the *GA-MinMin* and *GA-Sufferage* are at the same level.

The best values of *average resource utilization* metric were obtained by *GA-MinMin* in all instances of inconsistent ETC matrices. It is interesting to observe that the same meta-heuristic (i.e. *GA-MinMin*) gives also the worst results in resource utilization, in the case of consistent ETC matrices, where *GA-Sufferage* is the best in the case of low, and *GA-MaxMin* – in the case of high heterogeneity of resources. For semi-consistent ETC the best results for all but one instances are achieved by *GA-MaxMin*.

Most of the values of *standard deviation* ($s.d.$) are in the range of 0.8-10 %, which means that the proposed method should be improved in the future research to make the whole scheduling process more stable.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a two level Stackelberg game for modelling asymmetric users' behavior in Grid scheduling scenario. One user plays a Leader role on the first level. He is responsible for computing a planning of a large fraction of the tasks in the batch. The rest of the players, called Followers, act on the second level. They try to select the best strategy for the assignments of their tasks subject to Leader's strategy.

We defined this Stackelberg game as a hierarchical optimization problem solved by Genetic Algorithms (GAs) at Leader's level and by ad-hoc heuristics at Followers' level.

The results of the experimental study showed that the proposed approach achieved high values of the utilization metric. The best resource utilization values have been achieved by the *GA-MinMin* metaheuristic for the inconsistent ETC matrices. On the other hand, regarding the makespan and flowtime values we observed that in both cases it is *GA-MaxMin* which achieved the worst values. In the case of flowtime the best results are usually achieved by *GA-MinMin*. Finally we also noticed that values of makespan and flowtime are somehow worse than those reached by some GA-based schedulers in the case of just task execution

Table II
COMPARISON THE AVERAGE VALUES OF RESOURCE UTILIZATION, MAKESPAN AND FLOWTIME ACHIEVED BY THREE HYBRID META-HEURISTICS (IN ARBITRARY TIME UNITS; $s.d.$ - STANDARD DEVIATION)

	Average Resource Utilization ($\pm s.d.$)			Average Makespan ($\pm s.d.$)			Average Flowtime ($\pm s.d.$)		
Instance	**GA-MinMin**	**Ga-MaxMin**	**GA-Sufferage**	**GA-MinMin**	**Ga-MaxMin**	**GA-Sufferage**	**GA-MinMin**	**Ga-MaxMin**	**GA-Sufferage**
u_c_hihi	0.952 (±0.0953)	0.9714 (± 0.0345)	**0.9750** (±0.0262)	**8094910** (±600034)	8379870 (± 697710)	8289200 (±631202)	**1102560000** (±62551800)	1171190000 (± 108789000)	1138430000 (±95274900)
u_c_hilo	0.804 (±0.1488)	**0.9726** (±0.0653)	0.9646 (±0.0979)	178408 (±18836.8)	**163439** (±6533.07)	163488 (±10846.7)	29851300 (±1855790)	29901800 (±1572700)	**29702500** (±1761150)
u_c_lohi	0.9620 (±0.0975)	0.9746 (±0.0354)	**0.9747** (±0.0333)	**262745** (±19323.2)	274184 (±21665.6)	273984 (±24492.3)	**3661260** (±1662420)	3886100 (±4060500)	38464000 (±4362160)
u_c_lolo	0.7995 (±0.163233)	**0.9746** (±0.0572)	0.9711 (±0.0671)	6025.39 (±770.943)	5508.65 (±251.945)	**5490.56** (±240.11)	994443 (±51414.6)	997411 (±56050.1)	**991400** (±34862.6)
u_i_hihi	**0.9880** (±0.0264)	0.9878 (±0.0385)	0.987162 (± 0.0382)	**3543020** (±703298)	3579800 (±1102800)	3601000 (±888645)	**433603000** (±108287000)	442345000 (±190414000)	443630000 (±148015000)
u_i_hilo	**0.9967** (±0.0065)	0.9964 (±0.0085)	0.9956 (±0.0082)	79438.2 (±4562.55)	79499.7 (±4173.97)	**79246.**2 (±3295.87)	**13340900** (±749330)	13406600 (±894854)	13348700 (±680981)
u_i_lohi	**0.9895** (±0.0276)	0.987174 (±0.0302)	0.9884 (±0.0339)	126202 (±21982.9)	126349 (±18348.6)	**125665** (±24526.3)	**15400900** (±3503680)	15642300 (±3324080)	15401200 (±4401610)
u_i_lolo	**0.9966** (±0.0089)	0.9955 (±0.0087)	0.9963 (±0.0081)	**2752.27** (±133.731)	2771.87 (±149.972)	2757.03 (±168.561)	**465670** (±25385.4)	470380 (±34978.7)	467816 (±32745.6)
u_s_hihi	**0.9793** (±0.0285)	0.9781 (±0.0307)	0.9758 (±0.0593)	**4710420** (±578839)	4807480 (±663325)	4780570 (±853673)	598951000 (±105362000)	603485000 (±126013000)	**596179000** (±134002000)
u_s_hilo	0.9759 (±0.0689)	**0.9875** (±0.0201)	0.9856 (±0.0283)	103358 (±7.448.58)	103073 (±4526.48)	**102822** (±6191.47)	**17558200** (±1244710)	17737500 (±1078820)	17576400 (±1157260)
u_s_lohi	0.9763 (±0.0604)	**0.9772** (±0.0413)	0.9683 (±0.0617)	**143175** (±37307.5)	144396 (±44480.9)	151477 (±53702.9)	**18041500** (±4981350)	18301600 (±6657510)	19184900 (± 8207100)
u_s_lolo	0.9749 (±0.0741)	**0.9851** (±0.0257)	0.9843 (±0.0250)	3711.72 (±276.094)	3719.71 (±279.368)	**3703.41** (±218.364)	**636440** (±37980.7)	646408 (±59455.7)	640312 (±43452.1)

cost as objective (see [Xhafa et al., 2007 a)]), but our scenario is more realistic as it allows the introduction of hierarchical levels in Grid architecture. In our future work we will evaluate the proposed three meta-heuristics in a dynamic setting using a Grid simulator [Xhafa et al., 2007 b)].

ACKNOWLEDGMENT

Fatos Xhafa's research work is supported by General Secretariat of Universities, Ministry of Education, Spain.

REFERENCES

[Ali et al., 2000] S. Ali, H.J. Siegel, M. Maheswaran and D. Hensgen: "Task execution time modeling for heterogeneous computing systems", *Proceedings of Heterogeneous Computing Workshop* , 185–199, 2000. (HCW 2000)

[Baçsar and Olsder, 1995] T. Baçsar, and G. J. Olsder, 1995: *"Dynamic Nonco- operative Game Theory"*, Academic Press, London, second edition.

[Braun et al., 2001] T. D.Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund: "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", *Journal of Parallel and Distributed Computing*, 61(6):810-837, 2001.

[Garg et al., 2009] S.K. Garg, R. Buyya, and H.J. Segel: "Scheduling Parallel Aplications on Utility Grids: Time and Cost Trade-off Management", *In Proc. of the 32nd ACSC*, Wellington, Australia, 2009, CRPIT, Vol. 91, Bernard Mans Ed.

[Kwok et al., 2007] Y.-K. Kwok, K. Hwang and Sh. Song: "Selfish Grids: Game-Theoretic Modeling and NAS/PSA Benchmark Evaluation", *IEEE Transactions on Parallel and Distributing Systems*, Vol. 18 No. 5, 1-16, 2007.

[Roughgarden, 2001] T. Roughgarden: "Stackelberg Scheduling Strategies", *Proc. of STOC01*, July 6-8, 2001, Hersonissos, Crete, Greece.

[Xhafa et al., 2007 a)] F. Xhafa, J. Carretero and A. Abraham: "Genetic Algorithm Based Schedulers for Grid Computing Systems", *International Journal of Innovative Computing, Information and Control*, Vol. 3, No. 5, 1053-1071, 2007.

[Xhafa et al., 2007 b)] F. Xhafa, J. Carretero, L. Barolli and A. Durresi: "Requirements for an Event-Based Simulation Package for Grid Systems, *Journal of Interconnection Networks*, vol.8, No.2, 163–178, 2007, World Scientific Pub.

[Xhafa et al., 2007 c)] F. Xhafa, L. Barolli and A. Durresi: "Batch Mode Schedulers for Grid Systems", *International Journal of Web and Grid Services*, Vol. 3, No. 1, 19-37, 2007.